CMSCONSTRUCT

# IT Discovery Machine

## Data Transformation Module

## Report Workbook

Chris Satterthwaite

Jan 23, 2018

## Contents



## Document Revisions

| Author | Version | Date | Details |
|---|---|---|---|
| Chris Satterthwaite | 1.0 | 1/23/2019 | Document created |
| | | | |
| | | | |

# Introduction

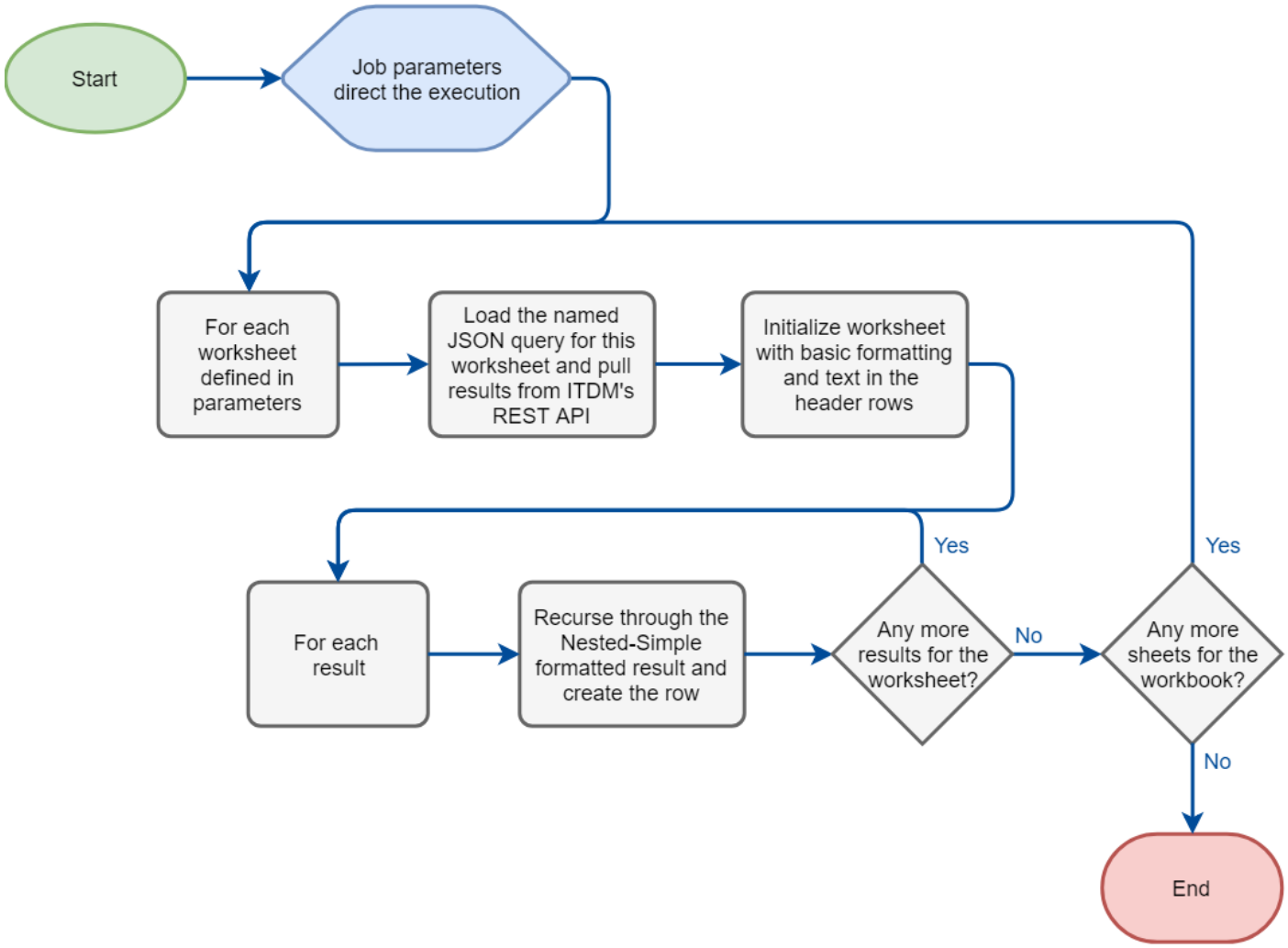
This document details the 'reportWorkbook' module written for IT Discovery Machine (ITDM).

Each job in this module generates an independent Excel workbook.

# Code Design

## Flow Diagram

The following diagram shows the reporting work flow:



# Pseudocode

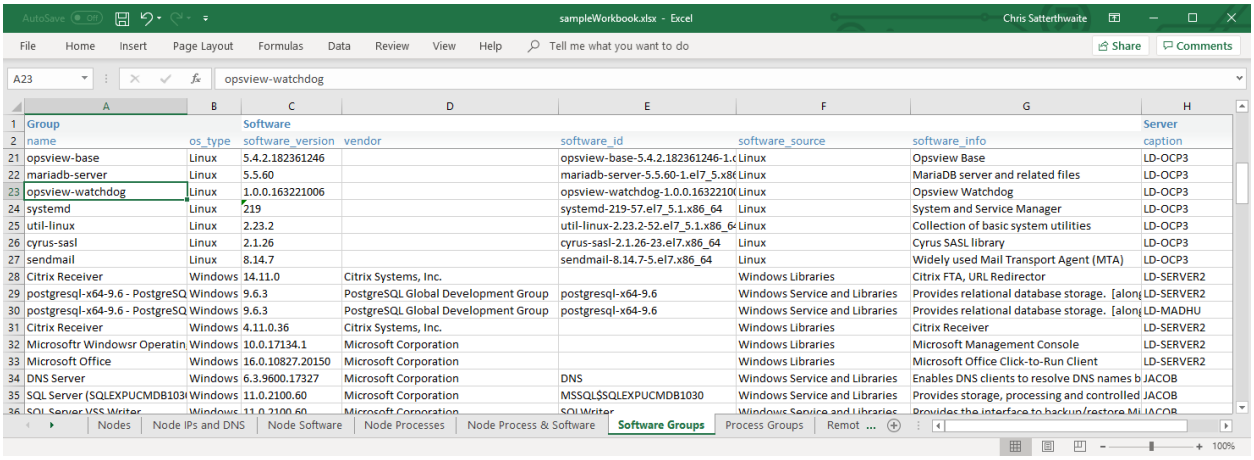
Pseudocode is an informal description of the design or algorithm of the code. This section takes the reader one level deeper into the working of the scripts. For specific functionality and code logic, please review the scripts themselves, which are written in Python with inline comments.

All jobs leverage the same design. Since this is a pretty basic module, the pseudocode looks similar to the flow diagram.

1. For each worksheet defined in the job's input parameters:
    a. Load the JSON query listed for the sheet. The queryName attribute will reference a file in the module's input directory, and so if the queryName is "nodes" then the module will have a file ./input/nodes.json.
    b. Initialize the worksheet with basic formatting, and fill in the header rows.
    c. For each result returned from the API:
        i. Recurse through the tree result (formatted Nested-Simple) and create the row
2. Finalize and write the workbook out to the module's runtime directory. The name of the file is based on the value of "reportFile" in the job's input parameter. So if a job has that set to "All_my_nodes", it will create the file ./runtime/All_my_nodes.xlsx.
3. Update the ITDM job status and exit.

# Sample Output

If you run the report_node_details job, you can browse sample output in the context of your environment. But here's a snapshot that illustrates the basic formatting of the header rows, sample data, and several worksheets in the same workbook (named tabs at the bottom):




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | Group | | Software | | | | | Server |
| 2 | name | os_type | software_version | vendor | software_id | software_source | software_info | caption |
| 21 | opsview-base | Linux | 5.4.2.182361246 | | opsview-base-5.4.2.182361246-1.( | Linux | Opsview Base | LD-OCP3 |
| 22 | mariadb-server | Linux | 5.5.60 | | mariadb-server-5.5.60-1.el7_5.x8( | Linux | MariaDB server and related files | LD-OCP3 |
| 23 | opsview-watchdog | Linux | 1.0.0.163221006 | | opsview-watchdog-1.0.0.1632210( | Linux | Opsview Watchdog | LD-OCP3 |
| 24 | systemd | Linux | 219 | | systemd-219-57.el7_5.1.x86_64 | Linux | System and Service Manager | LD-OCP3 |
| 25 | util-linux | Linux | 2.23.2 | | util-linux-2.23.2-52.el7_5.1.x86_6( | Linux | Collection of basic system utilities | LD-OCP3 |
| 26 | cyrus-sasl | Linux | 2.1.26 | | cyrus-sasl-2.1.26-23.el7.x86_64 | Linux | Cyrus SASL library | LD-OCP3 |
| 27 | sendmail | Linux | 8.14.7 | | sendmail-8.14.7-5.el7.x86_64 | Linux | Widely used Mail Transport Agent (MTA) | LD-OCP3 |
| 28 | Citrix Receiver | Windows | 14.11.0 | Citrix Systems, Inc. | | Windows Libraries | Citrix FTA, URL Redirector | LD-SERVER2 |
| 29 | postgresql-x64-9.6 - PostgreSQ | Windows | 9.6.3 | PostgreSQL Global Development Group | postgresql-x64-9.6 | Windows Service and Libraries | Provides relational database storage. [alon( | LD-SERVER2 |
| 30 | postgresql-x64-9.6 - PostgreSQ | Windows | 9.6.3 | PostgreSQL Global Development Group | postgresql-x64-9.6 | Windows Service and Libraries | Provides relational database storage. [alon( | LD-MADHU |
| 31 | Citrix Receiver | Windows | 4.11.0.36 | Citrix Systems, Inc. | | Windows Libraries | Citrix Receiver | LD-SERVER2 |
| 32 | Microsoftr Windowsr Operatin | Windows | 10.0.17134.1 | Microsoft Corporation | | Windows Libraries | Microsoft Management Console | LD-SERVER2 |
| 33 | Microsoft Office | Windows | 16.0.10827.20150 | Microsoft Corporation | | Windows Libraries | Microsoft Office Click-to-Run Client | LD-SERVER2 |
| 34 | DNS Server | Windows | 6.3.9600.17327 | Microsoft Corporation | DNS | Windows Service and Libraries | Enables DNS clients to resolve DNS names b | JACOB |
| 35 | SQL Server (SQLEXPUCMDB103( | Windows | 11.0.2100.60 | Microsoft Corporation | MSSQL$SQLEXPUCMDB1030 | Windows Service and Libraries | Provides storage, processing and controlled | JACOB |
| 36 | SQL Server VSS Writer | Windows | 11.0.2100.60 | Microsoft Corporation | SQLWriter | Windows Service and Libraries | Provides the interface to backup/restore Mi | JACOB |

# Module Contents

## Jobs

For general information on jobs (e.g. standard sections, parameter descriptions, general usage), refer to the Job Descriptor document.

Initially there are two (2) jobs available in this module:

| File name | Description |
|---|---|
| report_node_details.json | Creates a workbook with nodes, IPs, and different software and process contexts from dynamic discovery. |
| report_logical_models.json | Creates a workbook with application models and metadata. |

The main difference between job definitions is with the input parameters, since they direct the workbook creation. Let's say a job's input parameters looked like the following:

```
"inputParameters" : {
        "printDebug" : true,
        "reportFile" : "nodeDetails",
        "worksheets" : [
                {
                        "sheetName" : "Nodes",
                        "queryName" : "node"
                },{
                        "sheetName" : "Node IPs and DNS",
                        "queryName" : "node_ip_dns"
                },{
                        "sheetName" : "Node Software",
                        "queryName" : "node_software"
                }
        ]
}
```

That directs the script to create a workbook called "nodeDetails.xlsx". The first worksheet would be called "Nodes", filled with data from the "node.json" input query. The second worksheet would be called "Node IPs and DNS", filled with data from the "node_ip_dns.json" input query. And the third and final worksheet would be called "Node Software", filled with data from the "node_software.json" input query.

The list of definitions in the worksheets parameter is an ordered list. If we wanted to add a fourth worksheet, we would just add another to the end of the worksheets variable list, with a sheetName and queryName. If we wanted to insert a new worksheet and have it take the second tabbed position, we would insert it after the "Nodes" sheet. That way all other worksheets would just be moved one position down in order.

## Scripts

There is a single script with this module, directed by the job's input parameters:

| File name | Description |
|---|---|
| reportWorkbook.py | Called directly by the job |

## Input Queries

There are ten (10) input queries in this module; one per worksheet definition:

| File name | Description |
|---|---|
| application_metadata.json | Used by the report_logical_models job |
| application_models.json | Used by the report_logical_models job |
| node.json | Used by the report_node_details job |
| node_ip_dns.json | Used by the report_node_details job |
| node_process.json | Used by the report_node_details job |
| node_process_software.json | Used by the report_node_details job |
| node_software.json | Used by the report_node_details job |
| process_dependencies.json | Used by the report_node_details job |
| process_groups.json | Used by the report_node_details job |
| software_groups.json | Used by the report_node_details job |

These queries are sent to the REST API's /ocp/task resource. For more details on query construction and API usage, see the REST API Reference.

# User Guide

This module enables a reporting framework, meaning it will be used to create new/custom reporting jobs. To create a new report, you create a new job definition as well as the queries you want it to populate the report. The JSON job file is dropped in the module's job directory next to the initial two jobs, and the JSON queries for the worksheet content are dropped in the module's input directory.

We've provided information on the job structure above, now let's walk through an example of the JSON query to fill out a worksheet.

The contents of the node_software.json query file follows:

```
{
        "objects": [
                {
                        "label": "Server",
                        "class_name": "Node",
                        "attributes": ["caption"],
                        "minimum": "1",
                        "maximum": "",
                        "linchpin": true
                },
                {
                        "label": "Software",
                        "class_name": "SoftwareFingerprint",
                        "attributes": [ "name", "software_info", "software_version", "vendor"],
                        "minimum": "1",
                        "maximum": ""
                }
        ],
        "links": [
                {
                        "label": "Server_to_Software",
                        "class_name": "Enclosed",
                        "first_id": "Server",
                        "second_id": "Software"
                }
        ]
}
```

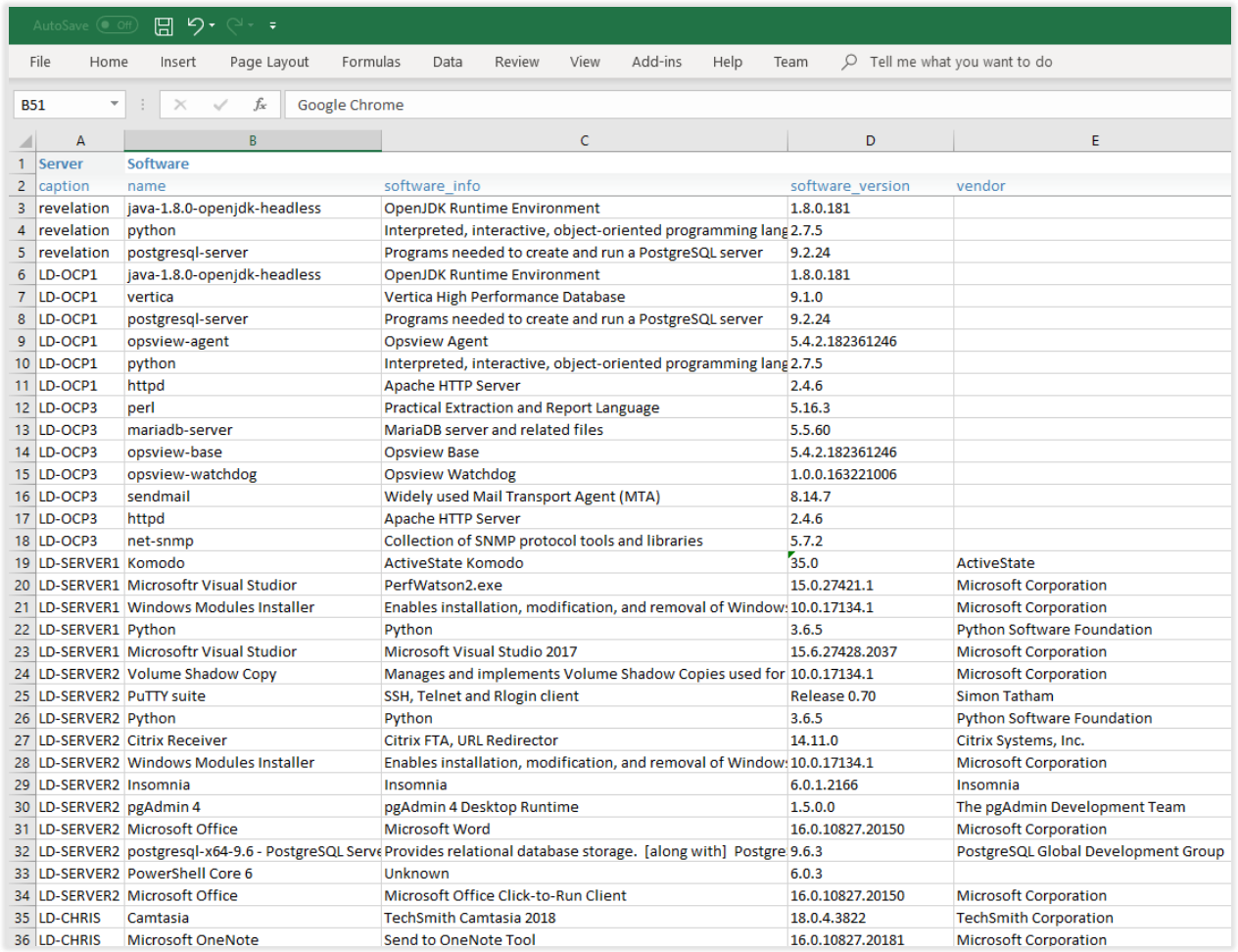
Comments:

- In the objects section, we see an ordered list of two Python classes:
    - The "Node" class which has the label "Server"
        - Only the "caption" attribute is requested for the report
        - There is a required minimum of 1 Node, with no maximum set
        - This query will start at the Node (since it has the linchpin)
    - The "SoftwareFingerprint" class which has the label "Software"

- There are four (4) attributes requested: “name”, “software_info”, “software_version”, “vendor”
- There is a required minimum of 1 SoftwareFingerprint, with no maximum set.

- In the links section, we see one link definition that connects “Server” to “Software”

The resulting Excel worksheet will look like the following snapshot:



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Server | Software | | | |
| 2 | caption | name | software_info | software_version | vendor |
| 3 | revelation | java-1.8.0-openjdk-headless | OpenJDK Runtime Environment | 1.8.0.181 | |
| 4 | revelation | python | Interpreted, interactive, object-oriented programming lang | 2.7.5 | |
| 5 | revelation | postgresql-server | Programs needed to create and run a PostgreSQL server | 9.2.24 | |
| 6 | LD-OCP1 | java-1.8.0-openjdk-headless | OpenJDK Runtime Environment | 1.8.0.181 | |
| 7 | LD-OCP1 | vertica | Vertica High Performance Database | 9.1.0 | |
| 8 | LD-OCP1 | postgresql-server | Programs needed to create and run a PostgreSQL server | 9.2.24 | |
| 9 | LD-OCP1 | opsview-agent | Opsview Agent | 5.4.2.182361246 | |
| 10 | LD-OCP1 | python | Interpreted, interactive, object-oriented programming lang | 2.7.5 | |
| 11 | LD-OCP1 | httpd | Apache HTTP Server | 2.4.6 | |
| 12 | LD-OCP3 | perl | Practical Extraction and Report Language | 5.16.3 | |
| 13 | LD-OCP3 | mariadb-server | MariaDB server and related files | 5.5.60 | |
| 14 | LD-OCP3 | opsview-base | Opsview Base | 5.4.2.182361246 | |
| 15 | LD-OCP3 | opsview-watchdog | Opsview Watchdog | 1.0.0.163221006 | |
| 16 | LD-OCP3 | sendmail | Widely used Mail Transport Agent (MTA) | 8.14.7 | |
| 17 | LD-OCP3 | httpd | Apache HTTP Server | 2.4.6 | |
| 18 | LD-OCP3 | net-snmp | Collection of SNMP protocol tools and libraries | 5.7.2 | |
| 19 | LD-SERVER1 | Komodo | ActiveState Komodo | 35.0 | ActiveState |
| 20 | LD-SERVER1 | Microsoftr Visual Studior | PerfWatson2.exe | 15.0.27421.1 | Microsoft Corporation |
| 21 | LD-SERVER1 | Windows Modules Installer | Enables installation, modification, and removal of Window | 10.0.17134.1 | Microsoft Corporation |
| 22 | LD-SERVER1 | Python | Python | 3.6.5 | Python Software Foundation |
| 23 | LD-SERVER1 | Microsoftr Visual Studior | Microsoft Visual Studio 2017 | 15.6.27428.2037 | Microsoft Corporation |
| 24 | LD-SERVER2 | Volume Shadow Copy | Manages and implements Volume Shadow Copies used for | 10.0.17134.1 | Microsoft Corporation |
| 25 | LD-SERVER2 | PuTTY suite | SSH, Telnet and Rlogin client | Release 0.70 | Simon Tatham |
| 26 | LD-SERVER2 | Python | Python | 3.6.5 | Python Software Foundation |
| 27 | LD-SERVER2 | Citrix Receiver | Citrix FTA, URL Redirector | 14.11.0 | Citrix Systems, Inc. |
| 28 | LD-SERVER2 | Windows Modules Installer | Enables installation, modification, and removal of Window | 10.0.17134.1 | Microsoft Corporation |
| 29 | LD-SERVER2 | Insomnia | Insomnia | 6.0.1.2166 | Insomnia |
| 30 | LD-SERVER2 | pgAdmin 4 | pgAdmin 4 Desktop Runtime | 1.5.0.0 | The pgAdmin Development Team |
| 31 | LD-SERVER2 | Microsoft Office | Microsoft Word | 16.0.10827.20150 | Microsoft Corporation |
| 32 | LD-SERVER2 | postgresql-x64-9.6 - PostgreSQL Serve | Provides relational database storage. [along with] Postgre | 9.6.3 | PostgreSQL Global Development Group |
| 33 | LD-SERVER2 | PowerShell Core 6 | Unknown | 6.0.3 | |
| 34 | LD-SERVER2 | Microsoft Office | Microsoft Office Click-to-Run Client | 16.0.10827.20150 | Microsoft Corporation |
| 35 | LD-CHRIS | Camtasia | TechSmith Camtasia 2018 | 18.0.4.3822 | TechSmith Corporation |
| 36 | LD-CHRIS | Microsoft OneNote | Send to OneNote Tool | 16.0.10827.20181 | Microsoft Corporation |

Notice it follows the order of the JSON query, starting at the linchpin. The columns are grouped by the defined classes and attributes are in the order requested in the query. So, first we have the linchpin object (Node) with its requested attributes (caption). Then we have the first connected object (Software), with its requested attributes (name, software_info, software_version, vendor). If there were more objects and attributes defined in the JSON query, the report would have more columns.